tions of the variables to produce shuffled stack locations; and updating the stack locations of the variables with the shuffled stack locations.

The identifying the stack locations may include identifying variables in object code, and the updating the stack locations of the variables may include updating the object code with the shuffled stack locations.

The identifying the stack locations may include identifying variables in binary code, and the updating the stack locations of the variables may include updating the binary code with the shuffled stack locations.

The identifying the stack locations may include identifying variables in executable code loaded into memory, and the updating the stack locations of the variables may include updating the executable code loaded into memory with the shuffled stack locations.

The shuffling the stack locations of the variables to produce shuffled stack locations and the updating the stack locations of the variables with the shuffled stack locations may occur prior to execution of the executable code.

The shuffling the stack locations of the variables to produce shuffled stack locations and the updating the stack locations of the variables with the shuffled stack locations may occur during execution of the executable code.

According to one embodiment of the present invention, a computer system includes a processor; and memory storing program instructions, the program instructions being configured to control the computer system to: identify a plurality of stack locations corresponding to a plurality of variables; shuffle the stack locations of the variables to produce shuffled stack locations; and update the stack locations of the variables with the shuffled stack locations.

The identifying the stack locations may include identifying variables in object code, and the updating the stack locations of the variables may include updating the object code with the shuffled stack locations.

The identifying the stack locations may include identifying variables in binary code, and the updating the stack locations of the variables may include updating the binary code with the shuffled stack locations.

The identifying the stack locations may include identifying variables in executable code loaded into memory, and the updating the stack locations of the variables may include updating the executable code loaded into memory with the shuffled stack locations.

The shuffling the stack locations of the variables to produce shuffled stack locations and the updating the stack locations of the variables with the shuffled stack locations may occur prior to execution of the executable code.

The shuffling the stack locations of the variables to produce shuffled stack locations and the updating the stack locations of the variables with the shuffled stack locations may occur during execution of the executable code.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, together with the specification, illustrate exemplary embodiments of the present invention, and, together with the description, serve to explain the principles of the present invention.

FIG. **1** is a schematic diagram of a typical computer system.

FIG. **2** is a schematic diagram illustrating a stack smashing buffer overflow attack.

FIG. **3** is a schematic diagram of an exemplary stack before and after being shuffled in accordance with an embodiment of the present invention.

FIG. **4** is a process flow diagram illustrating a method of randomizing the locations of variables on the stack during a standard build process in accordance with one embodiment of the present invention.

FIG. **5** is a process flow diagram illustrating a method of randomizing the locations of variables on the stack in a compiled and linked executable binary file (or binary) in accordance with one embodiment of the present invention.

FIG. **6** is a process flow diagram illustrating a method of randomizing the locations of variables on the stack after a compiled and linked executable binary file (or binary) is loaded into memory for execution in accordance with one embodiment of the present invention.

FIG. **7** is a process flow diagram illustrating a method of randomizing the locations of variables on the stack during execution in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

In the following detailed description, only certain exemplary embodiments of the present invention are shown and described, by way of illustration. As those skilled in the art would recognize, the invention may be embodied in many different forms and should not be construed as being limited to the embodiments set forth herein. Like reference numerals designate like elements throughout the specification.

Embodiments of the present invention are directed to methods, systems, and programs for improving the security of a computer system against code injection attacks by shuffling (or randomizing) the order of local function variables on the stack. Embodiments of the present invention may be implemented as a computer programmed in any of a number of programming languages such as x86 assembly, C, Python, and Java. Embodiments of the present invention may also be implemented as a computer program stored on a non-transitory computer readable medium embodying program instructions. Some embodiments of the present invention are directed to the method of performing the processes described below.

FIG. **3** is a schematic diagram of an exemplary stack before and after being shuffled in accordance with an embodiment of the present invention. Referring to FIG. **3**, prior to stack randomization, the stack **102** includes (in order from the bottom of the stack toward the top of the stack **112**): a return address, local variable 3, an attackable buffer, local variable 2, and local variable 1. In exemplary stack **102**, there is a critical distance (Cd_{before}) between the return address and the address of the attackable buffer, where critical distance is the sum of the size of the local variable 3 and the size of the attackable buffer.

Stack **104** in FIG. **3** shows the stack after stack randomization. In stack **104**, local variable 1 and local variable 2 are now located between the attackable buffer and the return address. As such, the critical distance (Cd_{after}) after shuffling is the sum of the size of local variables 1 and 2 and the attackable buffer. As such, the critical distance Cd between the attackable buffer and the return address is changed by the randomization process.

By randomizing this order within individual stack frames of a program, an attacker would not be able to know where an "attackable" buffer would be placed on the stack (the critical distance Cd), thereby providing additional security against stack-based exploits by frustrating or preventing an attacker's ability to craft an appropriate attack payload due to the uncertainty in the distance between the start of the buffer and the location of the return address on the stack. While code injec-